OBJECT ORIENTED PROGRAMMING LAB

ETCS-258

Faculty name: Ms. ZAMEER FATIMA

Student name: NIKHIL MATHUR

Roll No.: 05214802719

Semester: 4th

Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085.

Object Oriented Programming Lab
(ETCS -258)
Lab Assessment Sheet.

Student En. No.: 05214802719 **Student Name:** NIKHIL MATHUR

Lab No.	Experiment	Date	Marks					Total marks	Signature
			R1	R2	R3	R4	R5		
1	1. Write a program to find whether a number is prime or not using OOP. 2. Write a program for multiplication of two matrices using OOP. 3. Write a program to take name, address as character string, age as int, salary as float and contains inline function to set the values and display it.	15/03/21							
2	4. Using, the concept of function overloading, write function for calculating area of triangle, circle and rectangle. 5. Create a class Student, which have data members as name, branch, rollno, age, sex, five subjects. Display the name of student & his percentage who has more than 70%. 6. Write a program to generate a magic square using OOP.	22/03/21							
	7. Create a class Time with members hours, minutes, seconds. Take input, add two objects passing objects to function and display result. 8. Write a program to enter any number and find its factorial using constructor.								

3	9. Write a program to perform addition of two complex numbers using constructor overloading. The first constructor which takes no argument is used to create objects which are not initialized, second which takes one argument is used to initialize real and imag parts to equal values and third which takes two argument is used to initialize real and imag to two different values.	26/04/21							
	10. Write a program to generate a Fibonacci series using copy constructor.								
4	11. Create a class which keep track of number of its instances. Use static data member, constructors and destructors to maintain updated information about active objects.	03/05/21							
	12. Write a program to access members of a student class using pointer to object members (or using indirection operator).								
	13. Write a program to demonstrate the use of "this" pointer.								
5	14. Write a program to find the biggest of three numbers using friend function.	17/05/21							
	15. Write a program to demonstrate the use of friend function with inline assignment.								
	16. Write a program to find the greater of two given numbers in two different classes using friend function.								
	17. Write a program to find the sum of two numbers declared in a class and display the numbers and sum using friend class.								
	18. Write a program to overload unary increment (++) operator.								

6	19. Write a program to overload binary + operator.	24/05/21							
	20. Write a program to overload less than (<) operator.								
	21. Write a program to overload assignment (=) operator.								
7	22. Write a program to overload new and delete operator.	31/05/21							
	23. Write a program to overload unary minus(-) operator using friend function.								
	24. Create a base class basic_info with data members name ,roll no, sex and two member functions getdata and display. Derive a class physical_fit from basic_info which has data members height and weight and member functions getdata and display. Display all the information using object of derived class.								
8	25. Create class first with data members book no, book name and member function getdata() and putdata(). Create a class second with data members author name, publisher and members getdata() and showdata(). Derive a class third from first and second with data member no of pages and year of publication. Display all these information using array of objects of the third class.								

	26. Design three classes STUDENT , EXAM and RESULT. The STUDENT class has data members such as rollno, name. create a class EXAM by inheriting the STUDENT class. The EXAM class adds data members representing the marks scored in six subjects. Derive the RESULT from the EXAM class and has its own data members such as total marks. Write a program to model this relationship.	07/06/21							
	27. Create a base class called SHAPE. Use this class to store two double type values. Derive two specific classes called TRIANGLE and RECTANGLE from the base class. Add to the base class, a member function get data to initialize base class data members and another member function display to compute and display the area of figures. Make display a virtual function and redefine this function in the derived classes to suit their requirements. Using these three classes design a program that will accept driven of a TRINGLE or RECTANGLE interactively and display the area.								
	28. Create a class called LIST with two pure virtual function store() and retrieve(). To store a value call store and to retrieve call retrieve function, Derive two classes stack and queue from it and override store and retrieve.								

Overall Comments:

Faculty Name: Ms. Zameer Fatima

Signature

LAB-1

15-03-21

EXPERIMENT 1.

AIM : Write a programme to find whether a number is prime or not .

THEORY :

A number which is only divisible by itself and 1 is known as a prime number, for example: 5 is a prime number because it is only divisible by itself and 1.
This program takes the value of number (entered by user) and checks whether the number is prime number or not.

ALGORITHM :

START

Step 1 → Take integer variable A

Step 2 → Divide the variable A with (A-1 to 2)

Step 3 → If A is divisible by any value (A-1 to 2) it is not prime

Step 4 → Else it is prime

STOP

CODE :

```cpp
#include <iostream>
using namespace std;
int main(){
   int num;
   bool flag = true;
   cout<<"Enter any number(should be positive integer): ";
   cin>>num;

   for(int i = 2; i <= num / 2; i++) {
      if(num % i == 0) {
         flag = false;
         break;
      }
   }
   if (flag==true)
      cout<<num<<" is a prime number";
   else
      cout<<num<<" is not a prime number";
   return 0;
}
```

OUTPUT :

```
Enter any number(should be positive integer): 5
5 is a prime number

...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 2.

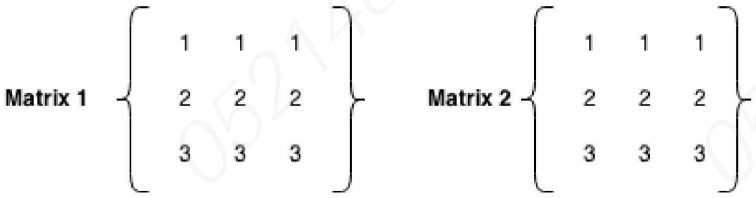
AIM : Write a program for multiplication of matrices using OOPS.

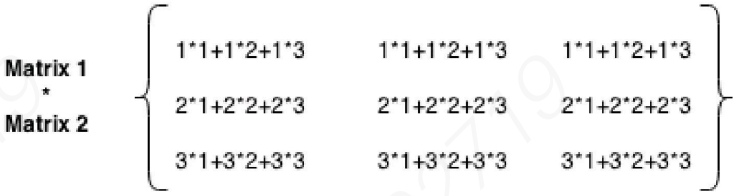
THEORY :

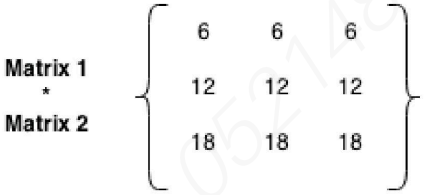
We can add, subtract, multiply and divide 2 matrices. To do so, we are taking input from the user for row number, column number, first matrix elements and second matrix elements. Then we are performing multiplication on the matrices entered by the user.

In matrix multiplication, the first matrix one row element is multiplied by the second matrix of all column elements.

Let's try to understand the matrix multiplication of 3*3 and 3*3 matrices by the figure given below:







CODE :

```cpp
#include <iostream>
using namespace std;
int main()
{
int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;
cout<<"enter the number of row=";
cin>>r;
cout<<"enter the number of column=";
cin>>c;
cout<<"enter the first matrix element=\n";
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
cin>>a[i][j];
}
}
cout<<"enter the second matrix element=\n";
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
cin>>b[i][j];
}
}
cout<<"multiply of the matrix=\n";
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
mul[i][j]=0;
for(k=0;k<c;k++)
{
mul[i][j]+=a[i][k]*b[k][j];
}
}
}
//for printing result
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
cout<<mul[i][j]<<" ";
}
cout<<"\n";
}
return 0;
}
```

OUTPUT :

```
enter the number of row=2
enter the number of column=2
enter the first matrix element=
1
2
3
4
enter the second matrix element=
1
2
3
4
multiply of the matrix=
7 10
15 22


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 3.

AIM : Write a program to take name, address as character string, age as int, salary as float and contains an inline function to set the values and display it.

THEORY:

String is a collection of characters. There are two types of strings commonly used in C++ programming language:

1. Strings that are objects of string class (The Standard C++ Library string class)
2. C-strings (C-style Strings)

C-strings :

In C programming, the collection of characters is stored in the form of arrays. This is also supported in C++ programming. Hence it's called C-strings. C-strings are arrays of type char terminated with null character, that is, \0 (ASCII value of null character is 0).

CODE :

```cpp
#include<iostream>
using namespace std;

inline void takeAndDisplay(string name,string address,int age,float salary)
{
    cout<<"\nThe details of the employee are\n"<<"Name: "<<name<<"\nAge: "<<age<<"\nAddress: "<<address<<"\nSalary: "<<salary<<endl;
}

int main()
{
    string name,address;
    int age;
    float salary;
    cout<<"Enter the name of the employee: ";
    cin>>name;
    cout<<"Enter the age of the employee: ";
    cin>>age;
    cout<<"Enter the address of the employee: ";
    cin>>address;
    cout<<"Enter the salary of the employee: ";
    cin>>salary;
    takeAndDisplay(name,address,age,salary);
    return 0;
}
```

OUTPUT :

```
Enter the name of the employee: NIKHIL
Enter the age of the employee: 20
Enter the address of the employee: DELHI
Enter the salary of the employee: 50K

The details of the employee are
Name: NIKHIL
Age: 20
Address: DELHI
Salary: 50


...Program finished with exit code 0
Press ENTER to exit console.
```

LAB-2

22-03-21

EXPERIMENT 4.

AIM : Write a program to find the area of triangle, circle and rectangle using the concept of function overloading.

THEORY:

Function Overloading in C++ :

Function overloading is a feature in C++ where two or more functions can have the same name but different parameters.

When a function name is overloaded with different jobs it is called Function Overloading.

In Function Overloading "Function" name should be the same and the arguments should be different.

Function overloading can be considered as an example of polymorphism feature in C++.

CODE :

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void area(float a,float b,float c)
{
     float s,area;
     s=(a+b+c)/2;
     area=sqrt(s*(s-a)*(s-b)*(s-c));
     cout<<"AREA="<<area<<" square units";
}
void area(float r)
{
     float area;
     area=3.14*r*r;
     cout<<"AREA="<<area<<" square units";
}
void area(float l,float b)
{
     float area;
     area=l*b;
     cout<<"AREA="<<area<<" square units";
}
void main()
{
     clrscr();
     float a,b,c;
     int ch;
     cout<<"1.Area of Triangle"<<endl;
```

```
29        cout<<"2.Area of Circle"<<endl;
30        cout<<"3.Area of Rectangle"<<endl;
31        cout<<"Enter choice:";
32        cin>>ch;
33        switch(ch)
34        {
35         case 1:cout<<"Enter first side:";
36            cin>>a;
37            cout<<"Enter second side:";
38            cin>>b;
39            cout<<"Enter third side:";
40            cin>>c;
41            area(a,b,c);
42            break;
43         case 2:cout<<"Enter radius:";
44            cin>>a;
45            area(a);
46            break;
47         case 3:cout<<"Enter length:";
48            cin>>a;
49            cout<<"Enter breadth:";
50            cin>>b;
51            area(a,b);
52            break;
53         default:cout<<"Wrong choice";
54        }
55        getch();
56  }
```

OUTPUT :

Running Turbo C Project

```
1.Area of Triangle
2.Area of Circle
3.Area of Rectangle
Enter choice:3
Enter length:12
Enter breadth:10
AREA=120 square units
```

EXPERIMENT 5.

AIM : Create a class Student, which have data members as name, branch, rollno, age, sex, five subjects. Display the name of the student & his percentage who has more than 70%.

THEORY:

C++ Class Definitions :

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows −

```
class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
```

The keyword public determines the access attributes of the members of the class that follows it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as private or protected which we will discuss in a sub-section.

CODE :

```cpp
#include<iostream>
#include<vector>
using namespace std;

class student
{
private:
    string name,branch;
    int rollno,age;
    char sex;
    vector<int> subs;
public:
    void takeinput()
    {
        cout<<"Enter the name of the student: ";
        cin>>name;
        cout<<"Enter the rollno of the student: ";
        cin>>rollno;
        cout<<"Enter the age of the student: ";
        cin>>age;
        cout<<"Enter the sex of the student: ";
        cin>>sex;
        cout<<"Enter the branch of the student: ";
        cin>>branch;
        cout<<"Enter the marks in 5 subs: ";
        int k;
        for(int i=0;i<5;i++)
        {
            cin>>k;
            subs.push_back(k);
        }
    }
    float avg()
    {
        float sum=0;
        for(int i=0;i<5;i+=1)
            sum+=subs[i];
        return sum/5;
    }
    string retname()
    {
        return name;
    }
};

int main()
{
    int n;
    cout<<"Enter the number of students: ";
    cin>>n;
    vector<student> students(n);
    cout<<"Enter the details of each student: \n";
    for(int i=0;i<n;i++)
        students[i].takeinput();
    for(int i=0;i<n;i++)

        if(students[i].avg()> 70)
        {
            cout<<"Student: "<<students[i].retname()<<" With Percentage "<<students[i].avg()<<"%"<<endl;
        }
}
```

OUTPUT :

```
Enter the number of students: 2
Enter the details of each student:
Enter the name of the student: NIKHIL
Enter the rollno of the student: 4
Enter the age of the student: 20
Enter the sex of the student: M
Enter the branch of the student: CSE
Enter the marks in 5 subs: 70
70
70
70
70
Enter the name of the student: ROHIT
Enter the rollno of the student: 5
Enter the age of the student: 21
Enter the sex of the student: M
Enter the branch of the student: CSE
Enter the marks in 5 subs: 80
80
80
80
80
Student: ROHIT With Percentage 80%


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 6.

AIM : Write a program to create a class TIME with members hours, minutes and seconds. Take input, add two objects by passing them to function and display the result.

THEORY :

Define C++ Objects :

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box −

```
Box Box1;        // Declare Box1 of type Box
Box Box2;        // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

CODE :

```
#include<iostream.h>
#include<conio.h>
class time
{
private:
int hr;
int min;
int sec;
public:
void input();
void output();
void add(time t1,time t2);
};
void time::input()
{
cout<<"Enter number of hours:";
cin>>hr;
cout<<"Enter number of minutes:";
cin>>min;
cout<<"Enter number of seconds:";
cin>>sec;
}
void time::output()
{
cout<<"Hours:"<<hr<<endl;
cout<<"Minutes:"<<min<<endl;
cout<<"Seconds:"<<sec<<endl;
}
```

```
29  void time::add(time t1,time t2)
30  {
31  sec=t1.sec+t2.sec;
32  min=0;
33  hr=0;
34  while(sec>60)
35  {
36  min+=1;
37  sec-=60;
38  }
39  min+=t1.min+t2.min;
40  while(min>60)
41  {
42  hr+=1;
43  min-=60;
44  }
45  hr+=t1.hr+t2.hr;
46  }
47  void main()
48  {
49  clrscr();
50  time t1,t2,t3;
51  cout<<"Enter T1==>"<<endl;
52  t1.input();
53  cout<<"\nEnter T2==>"<<endl;
54  t2.input();
55  t3.add(t1,t2);
56  cout<<"\nSum of T1 and T2==>"<<endl;
57  t3.output();
58  getch();
59  }
```

OUTPUT :

Running Turbo C Project

```
Enter T1==>
Enter number of hours:
10
Enter number of minutes:2
Enter number of seconds:10

Enter T2==>
Enter number of hours:20
Enter number of minutes:4
Enter number of seconds:20

Sum of T1 and T2==>
Hours:30
Minutes:6
Seconds:30
```

LAB-3

26-04-21

EXPERIMENT 7.

AIM : Write a program to access members of a student class using pointer to object members(or using indirection operation).

THEORY :

C++ Class Definitions :

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object. A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows −

```
class Box {
  public:
    double length;   // Length of a box
    double breadth;  // Breadth of a box
    double height;   // Height of a box
};
```

The keyword public determines the access attributes of the members of the class that follows it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as private or protected which we will discuss in a sub-section.

Define C++ Objects :

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box −

```
Box Box1;        // Declare Box1 of type Box
Box Box2;        // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

CODE :

```cpp
#include<iostream>
#include<limits>
#include<string>
using namespace std;

class Student{
    int rollNo;
    string name;
    int year;
    string branch;
    public:
        Student(){
            cout<<"Student object created"<<endl;
        }
        void setData(){

            cout<<"\nEnter your roll no: ";
            cin>>rollNo;
            cout<<"\nEnter your name: ";
            cin.ignore(numeric_limits<streamsize>::max(),'\n');
            getline(cin,name);
            cout<<"\nEnter your branch: ";
            cin>>branch;
            cout<<"\nEnter your year: ";
            cin>>year;
            cout<<"\nDetails updated!!!!"<<endl;
        }

        void getData(){
            cout<<"\nStudent details:\nStdent name: "
            <<name<<"\nStdent roll no: "<<rollNo
            <<"\nStdent year: "<<year<<"\nStdent branch: "<<branch<<endl;
        }
};
int main(){
    Student*pointer;

    pointer = new Student();
    pointer->setData();
    pointer->getData();
    return 0;
}
```

OUTPUT :

```
Student object created

Enter your roll no: 5

Enter your name: NIKHIL MAHTUR

Enter your branch: CSE

Enter your year: 2019

Details updated!!!!

Student details:
Stdent name: NIKHIL MAHTUR
Stdent roll no: 5
Stdent year: 2019
Stdent branch: CSE


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 8.

AIM : Write a program to generate magic square using OOPS.

THEORY :

Magic Square :

The magic square is a square matrix, whose order is odd and where the sum of the elements for each row or each column or each diagonal is same.

8	1	6
3	5	7
4	9	2

The sum of each row or each column or each diagonal can be found using this formula. n(n2+ 1)/2

Here are the rules to construct a magic square −

- We will start from the middle column of the first row, of the matrix, and always go to the top left corner to place next number
- If the row exceeds, or the row is not in the matrix, then, change the column as left column and place the number at last row of the matrix, and go for top left corner again.
- If the column exceeds, or the column is not in the matrix, then change the row as top row and place the number at last column of that matrix, then go to the top left corner again.
- When the top left corner is not vacant or both row and column exceeds the range, then place the number at the bottom of the last-placed number.

CODE :

```cpp
#include<iostream>
#include<iomanip>
using namespace std;

void createSquare(int **array, int r, int c) {
   int i, j, count = 1, range;
   for(i = 0; i<r; i++)
      for(j = 0; j<c; j++)
         array[i][j] = 0;

   range = r*c;
   i = 0;
   j = c/2;
   array[i][j] = count;

   while(count < range) {
      count++;
      if((i-1) < 0 && (j-1) < 0)
         i++;
      else if((i-1) <0) {
         i = r-1;
         j--;
      }else if((j-1) < 0) {
         j = c-1;
         i--;
      }else if(array[i-1][j-1] != 0)
         i++;
      else{
         i--;
         j--;
      }
      array[i][j] = count;
   }

   for(i = 0; i<r; i++) {
      for(j = 0; j<c; j++)
         cout <<setw(3) << array[i][j];
      cout << endl;
   }
}

main() {
   int** matrix;
   int row, col;
   cout << "Enter the order(odd) of square matrix :";
   cin >> row;
   col = row;

   matrix = new int*[row];

   for(int i = 0; i<row; i++) {
      matrix[i] = new int[col];
   }
   createSquare(matrix, row, col);
}
```

OUTPUT :

```
Enter the order(odd) of square matrix :3
  6  1  8
  7  5  3
  2  9  4


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 9.

AIM : Write a program to enter any number and find its factorial using the constructor.

THEORY :

C++ Constructor :

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

There can be two types of constructors in C++.

- Default constructor - A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

- Parameterized constructor - A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

CODE :

```
#include<iostream.h>
#include<conio.h>
class copy
{
     private:
      int var,fact;
     public:
      copy(int temp)
      {
    var = temp;
      }
      double calculate()
      {
            fact=1;
            for(int i=1;i<=var;i++)
            {
             fact = fact * i;
            }
            return fact;
      }
};
void main()
{
     clrscr();
     int n;
cout<<"Enter the Number : ";
     cin>>n;
     copy obj(n);
     copy cpy=obj;
cout<<"Factorial of "<<n<<" is:"<<obj.calculate()<<endl;
cout<<"Factorial of "<<n<<" is:"<<cpy.calculate();
     getch();
}
```

OUTPUT :

Running Turbo C Project

```
Enter the Number : 5
Factorial of 5 is:120
Factorial of 5 is:120
```

EXPERIMENT 10.

AIM : Write a program to perform addition of two complex numbers using constructor overloading. The first constructor which takes no argument is used to create objects which are not initialized, second which takes one argument is used to initialize real and imaginary parts to equal value and third which takes two arguments is used to initialize real and imaginary parts to two different values.

THEORY :

Constructor Overloading in C++ :

In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments.This concept is known as Constructor Overloading and is quite similar to function overloading.

- Overloaded constructors essentially have the same name (name of the class) and different number of arguments.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

CODE :

```
#include<iostream.h>
#include<conio.h>
class complex
{
     private:
      double real;
      double imag;
     public:
      complex();
      complex(double a);
      complex(double x,double y);
      void add(complex c1,complex c2);
      void show();
};
complex::complex()
{
     real=0.0;
imag=0.0;
}
complex::complex(double a)
{
     real=a;
imag=a;
}
complex::complex(double x,double y)
{
     real=x;
imag=y;
}
void complex::add(complex c1,complex c2)
```

```
{
     real=c1.real+c2.real;
imag=c1.imag+c2.imag;
}
void complex::show()
{
     if(imag>0)
     {
    cout<<real<<"+"<<imag<<"i"<<endl;
     }
     else
     {
    cout<<real<<imag<<"i"<<endl;
     }
}
void main()
{
     clrscr();
     complex c1(6,8);
     complex c2(5);
     complex c3;
     c3.add(c1,c2);
cout<<"First complex number==>"<<endl;
     c1.show();
cout<<"Second complex number==>"<<endl;
     c2.show();
cout<<"Sum==>"<<endl;
     c3.show();
     getch();
}
```

OUTPUT :

Running Turbo C Project

```
First complex number==>
6+8i
Second complex number==>
5+5i
Sum==>
11+13i
```

LAB-4

03-05-21

EXPERIMENT 11.

AIM : Write a program to generate a Fibonacci series using copy constructor.

THEORY :

Copy Constructor in C++ :

A copy constructor is a member function that initializes an object using another object of the same class. A copy constructor has the following general function prototype:

ClassName (const ClassName &old_obj);

CODE :

```
#include<iostream.h>
#include<conio.h>
class fibonacci
{
     private:
      long int f0,f1,fib;
     public:
     fibonacci()
      {
            f0=0;
            f1=1;
            fib=f0+f1;
      }
     fibonacci(fibonacci&ptr)
      {
            f0=ptr.f0;
            f1=ptr.f1;
            fib=ptr.fib;
      }
      void increment()
      {
            f0=f1;
            f1=fib;
            fib=f0+f1;
      }
      void display()
      {
     cout<<fib<<'\t';
```

```
29        }
30  };
31  void main()
32  {
33        clrscr();
34        int n;
35  fibonacci number;
36  cout<<"Enter number of elements:";
37        cin>>n;
38        for(int i=1;i<=n;i++)
39        {
40      number.display();
41      number.increment();
42        }
43        getch();
44  }
```

OUTPUT :

Running Turbo C Project

```
Enter number of elements:8
1       2       3       5       8       13      21      34
```

EXPERIMENT 12.

AIM : Create a class which keeps track of the number of its instances. Use static data members,constructors and deconstructors to maintain updated information about active objects.

THEORY :

Static data members in C++ :

Static data member are class members that are declared using static keyword A static member has certain special characteristics These are:

- Only one copy of that member is created for the entire class and is shared by all the objects of that class , no matter how many objects are created.
- It is initialized to zero when the first object of its class is created .No other initialization is permitted
- It is visible only within the class,but its lifetime is the entire program

Syntax : static data_type data_member_name;

CODE :

```
#include<iostream>
using namespace std;

class Instance{
    public:
    static int object_count;
    Instance(){
        object_count++;
        cout<<"New instance created!!"<<endl;
        cout<<"Number of instance increased to "<<object_count<<endl;
    }
    ~Instance(){
        object_count--;
        cout<<"One Instance deleted!!"<<endl;
        cout<<"Number of instance decreased to "<<object_count<<endl;
        if(object_count==0){
            cout<<"There are no objects left, ending the progam"<<endl;
        }
    }
};

int Instance::object_count = 0;
int main(){
    Instance obj1;
    Instance obj2;
    Instance obj3;
    Instance obj4;
    Instance obj5;
    return 0;
}
```

OUTPUT :

```
New instance created!!
Number of instance increased to 1
New instance created!!
Number of instance increased to 2
New instance created!!
Number of instance increased to 3
New instance created!!
Number of instance increased to 4
New instance created!!
Number of instance increased to 5
One Instance deleted!!
Number of instance decreased to 4
One Instance deleted!!
Number of instance decreased to 3
One Instance deleted!!
Number of instance decreased to 2
One Instance deleted!!
Number of instance decreased to 1
One Instance deleted!!
Number of instance decreased to 0
There are no objects left, ending the progam


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 13.

AIM : Write a program to demonstrate use of "this" pointer.

THEORY :

'This' pointer :

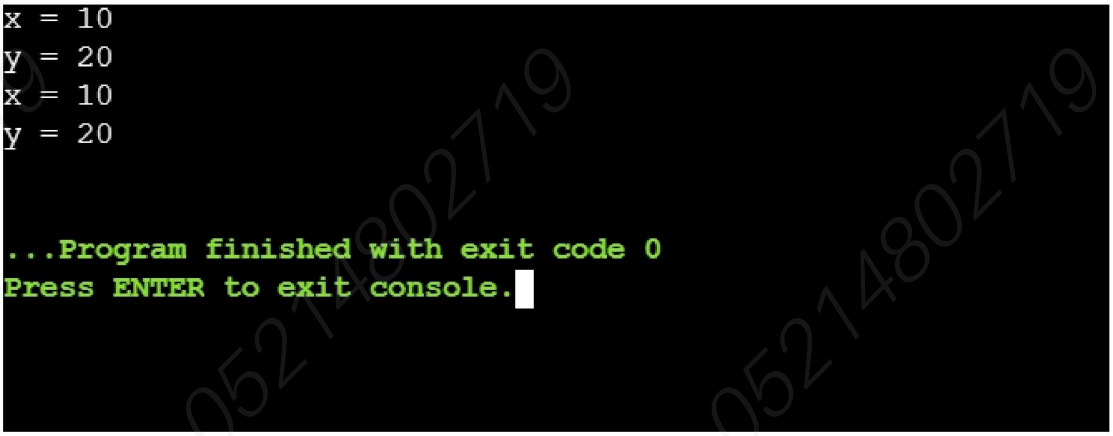
Every object in C++ has access to its own address through an important pointer called this pointer. The this pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a this pointer, because friends are not members of a class. Only member functions have a this pointer.

CODE :

```cpp
#include <iostream>
using namespace std;
class A
{
    private:
        int x;
        int y;
    public:
        A(int x, int y)
        {
            this->x = x;
            this->y = y;
        }
        void display()
        {
            cout<<"x = "<<x<<endl;
            cout<<"y = "<<y<<endl;
        }
        A& clone()
        {
            return *this;
        }
};
int main()
{
    A obj1(10, 20);
    obj1.display();
    A obj2 = obj1.clone();
    obj2.display();
    return 0;
}
```

OUTPUT :


x = 10
y = 20
x = 10
y = 20
...Program finished with exit code 0
Press ENTER to exit console.

LAB-5

17-05-21

EXPERIMENT 14.

AIM : Write a program to find the biggest of three numbers using the friend function.

THEORY :

Friend Function Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:
a) A member of another class
b) A global function

CODE :

```
#include<iostream.h>
#include<conio.h>
class largest
{
     private:
      int x,y,z;
     public:
      void num()
      {
    cout<<"Enter the first number:";
          cin>>x;
    cout<<"Enter the second number:";
          cin>>y;
    cout<<"Enter the third number:";
          cin>>z;
      }
      friend void max(largest l);
};
void max(largest l)
{
     if((l.x>l.y)&&(l.x>l.z))
     {
    cout<<"Largest="<<l.x<<endl;
     }
     else if((l.y>l.x)&&(l.y>l.z))
     {
    cout<<"Largest="<<l.y<<endl;
     }
```

```
29        else
30        {
31       cout<<"Largest="<<l.z<<endl;
32        }
33  }
34  void main()
35  {
36        clrscr();
37        largest l;
38  l.num();
39        max(l);
40        getch();
41  }
```

OUTPUT :

Running Turbo C Project

```
Enter the first number:13
Enter the second number:133
Enter the third number:1333
Largest=1333
```

EXPERIMENT 15.

AIM : Write a program to demonstrate the use of a friend function with inline assignment.

THEORY :

Inline Friend Function :

The friend functions, may also be declared as inline functions. If any friend function is declared inside the class or within the scope of the class definition, then the inline code substitution is automatically assigned to it. But if the friend function id defined outside the scope of class definition, then it is required to assign a keyword inline before the return type. The following example shows the assignment of inline to a friend function.

CODE :

```
#include<iostream.h>
#include<conio.h>
class Humidity;
class Temperature
{
     private:
      int m_temp;
     public:
      Temperature(int temp=0)
      {
        m_temp=temp;
      }
      void setTemperature(int temp)
      {
        m_temp=temp;
      }
      friend void printWeather(Temperature &temperature,Humidity&humidity);
};
class Humidity
{
     private:
      int m_humidity;
     public:
      Humidity(int humidity=0)
      {
        m_humidity=humidity;
      }
      void setHumidity(int humidity)
      {
        m_humidity=humidity;
      }
      friend void printWeather(Temperature &temperature,Humidity&humidity)
      {
        cout<<"The temperature is "<<temperature.m_temp<<" and the humidity is "<<humidity.m_humidity<<endl;
      }
};
void main()
{
    clrscr();
    Humidity hum(10);
    Temperature temp(12);
    printWeather(temp, hum);
    getch();
}
```

OUTPUT :

Running Turbo C Project

```
The temperature is 12 and the humidity is 10
```

EXPERIMENT 16.

AIM : Write a program to find the greatest of two given numbers in two different classes using a friend function.

THEORY :

Friend Function Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:
a) A member of another class
b) A global function

CODE :

```
#include<iostream.h>
#include<conio.h>
class B;
class A
{
     private:
      int x;
     public:
      void num()
      {
    cout<<"Enter number:";
          cin>>x;
      }
      friend void max(A a,B b);
};
class B
{
     private:
      int x;
     public:
      void num()
      {
    cout<<"Enter number:";
          cin>>x;
      }
      friend void max(A a,B b);
};
void max(A a,B b)
```

```
{
     if(a.x>b.x)
     {
    cout<<"Largest:"<<a.x<<endl;
     }
     else
     {
    cout<<"Largest:"<<b.x<<endl;
     }
}
void main()
{
     clrscr();
     A a;
     B b;
a.num();
b.num();
     max(a,b);
     getch();
}
```

OUTPUT :

Running Turbo C Project

```
Enter number:999
Enter number:99
Largest:999
```

EXPERIMENT 17.

AIM : Write a program to find the sum of two numbers declared in a class and display the numbers and sum using the friend function.

THEORY :

Friend Function Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:
a) A member of another class
b) A global function

CODE :

```
#include <iostream.h>
#include <conio.h>
class A
{
     private:
      int x,y;
     public:
      void num()
      {
    cout<<"Enter the first number:";
          cin>>x;
    cout<<"Enter the second number:";
          cin>>y;
      }
      friend void sum(A a);
};
void sum(A a)
{
cout<<"First number:"<<a.x<<endl;
cout<<"Second number:"<<a.y<<endl;
cout<<"Their sum:"<<a.x+a.y<<endl;
}
void main()
{
    clrscr();
    A a;
a.num();
    sum(a);
    getch();
}
```

OUTPUT :

Running Turbo C Project

```
Enter the first number:21
Enter the second number:22
First number:21
Second number:22
Their sum:43
```

LAB-6

24-05-21

EXPERIMENT 18.

AIM : Write a program to overload unary increment(++).

THEORY :

Overloading the increment operator :

The operator symbol for both prefix(++i) and postfix(i++) are the same. Hence, we need two different function definitions to distinguish between them. This is achieved by passing a dummy int parameter in the postfix version.

CODE :

```
#include<iostream.h>
#include<conio.h>
class overloading
{
     private:
     int a,b,c,d;
     public:
      void input()
      {
    cout<<"Enter the value of a:";
          cin>>a;
    cout<<"Enter the value of b:";
          cin>>b;
    cout<<"Enter the value of c:";
          cin>>c;
    cout<<"Enter the value of d:";
          cin>>d;
      }
      void output()
      {
    cout<<"Value of a:"<<a<<endl;
    cout<<"Value of b:"<<b<<endl;
    cout<<"Value of c:"<<c<<endl;
    cout<<"Value of d:"<<d<<endl;
      }
      void operator ++()
      {
          a++;
```

```
29              b++;
30          cout<<"Incremented value of a:"<<a<<endl;
31          cout<<"Incremented value of b:"<<b<<endl;
32            }
33            void operator --()
34            {
35                  c--;
36                  d--;
37          cout<<"Decremented value of c:"<<c<<endl;
38          cout<<"Decremented value of d:"<<d<<endl;
39            }
40  };
41  void main()
42  {
43          clrscr();
44          overloading o;
45  o.input();
46  o.output();
47          o++;
48          o--;
49          getch();
50  }
```

OUTPUT :

Running Turbo C Project

```
Enter the value of a:2
Enter the value of b:3
Enter the value of c:4
Enter the value of d:5
Value of a:2
Value of b:3
Value of c:4
Value of d:5
Incremented value of a:3
Incremented value of b:4
Decremented value of c:3
Decremented value of d:4
```

EXPERIMENT 19.

AIM : Write a program to overload binary + operator.

THEORY :

A binary operator is an operator that operates on two operands and manipulates them to return a result. Operators are represented by special characters or by keywords and provide an easy way to compare numerical values or character strings.
Binary operators are presented in the form:
Operand1 Operator Operand2

CODE :

```cpp
#include <iostream>
using namespace std;

class Sum {
   private:
    int num;

   public:
    Sum() : num(0) {}

    void input() {
        cout << "Enter the number: ";
        cin >> num;
    }

    Sum operator + (const Sum& obj) {
        Sum temp;
        temp.num = num + obj.num;
        return temp;
    }

    void output() {
      cout << "Output: " << num;
    }
};

int main() {
    Sum S1, S2, result;
    S1.input();
    S2.input();

    result = S1 + S2;
    result.output();
    return 0;
}
```

OUTPUT :

```
Enter the number: 22
Enter the number: 33
Output: 55

...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 20.

AIM : Write a program to overload less than(<) operator.

THEORY :

A binary operator is an operator that operates on two operands and manipulates them to return a result. Operators are represented by special characters or by keywords and provide an easy way to compare numerical values or character strings.
Binary operators are presented in the form:
Operand1 Operator Operand2

CODE :

```
#include <iostream>
using namespace std;

class Distance {
   private:
      int feet;
      int inches;

   public:

      Distance() {
         feet = 0;
         inches = 0;
      }
      Distance(int f, int i) {
         feet = f;
         inches = i;
      }


      void displayDistance() {
         cout << "F: " << feet << " I:" << inches <<endl;
      }


      Distance operator- () {
         feet = -feet;
         inches = -inches;
```

```
29          return Distance(feet, inches);
30       }
31
32
33       bool operator <(const Distance& d) {
34          if(feet < d.feet) {
35             return true;
36          }
37          if(feet == d.feet && inches < d.inches) {
38             return true;
39          }
40
41          return false;
42       }
43 };
44
45 int main() {
46    Distance D1(11, 10), D2(5, 11);
47
48    if( D1 < D2 ) {
49       cout << "D1 is less than D2 " << endl;
50    } else {
51       cout << "D2 is less than D1 " << endl;
52    }
53
54    return 0;
55 }
```

OUTPUT :

```
D2 is less than D1


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 21.

AIM : Write a program to overload assignment(=) operator.

THEORY :

You can overload the assignment operator (=) just as you can other operators and it can be used to create an object just like the copy constructor.

CODE :

```
#include <iostream>
using namespace std;

class Distance {
   private:
      int feet;
      int inches;

   public:
      Distance() {
         feet = 0;
         inches = 0;
      }
      Distance(int f, int i) {
         feet = f;
         inches = i;
      }
      void operator = (const Distance &D ) {
         feet = D.feet;
         inches = D.inches;
      }

      void displayDistance() {
         cout << "F: " << feet <<  " I:" <<  inches << endl;
      }
};
```

```
28 int main() {
29     Distance D1(11, 10), D2(5, 11);
30 
31     cout << "First Distance : ";
32     D1.displayDistance();
33     cout << "Second Distance :";
34     D2.displayDistance();
35 
36     D1 = D2;
37     cout << "First Distance :";
38     D1.displayDistance();
39 
40     return 0;
41 }
```

OUTPUT :

```
First Distance : F: 11 I:10
Second Distance :F: 5 I:11
First Distance :F: 5 I:11


...Program finished with exit code 0
Press ENTER to exit console.
```

LAB-7

31-05-21

EXPERIMENT 22.

AIM : Write a program to overload new and delete operator.

THEORY :

Overloading New and Delete operator in C++ :
The new and delete operators can also be overloaded like other operators in C++. New and Delete operators can be overloaded globally or they can be overloaded for specific classes.
If these operators are overloaded using member function for a class, it means that these operators are overloaded only for that specific class.
If overloading is done outside a class (i.e. it is not a member function of a class), the overloaded 'new' and 'delete' will be called anytime you make use of these operators (within classes or outside classes). This is global overloading.

Syntax for overloading the new operator :
void* operator new(size_t size);

The overloaded new operator receives size of type size_t, which specifies the number of bytes of memory to be allocated. The return type of the overloaded new must be void*.The overloaded function returns a pointer to the beginning of the block of memory allocated.

Syntax for overloading the delete operator :
void operator delete(void*);

The function receives a parameter of type void* which has to be deleted. Function should not return anything.
NOTE: Both overloaded new and delete operator functions are static members by default. Therefore, they don't have access to this pointer .

CODE :

```cpp
#include<iostream>
#include<stdlib.h>

using namespace std;
class student
{
    string name;
    int age;
public:
    student()
    {
        cout<< "Constructor is called\n" ;
    }
    student(string name, int age)
    {
        this->name = name;
        this->age = age;
    }
    void display()
    {
        cout<< "Name:" << name << endl;
        cout<< "Age:" << age << endl;
    }
    void * operator new(size_t size)
    {
        cout<< "Overloading new operator with size: " << size << endl;
        void * p = ::operator new(size);
        //void * p = malloc(size); will also work fine

        return p;
    }

    void operator delete(void * p)
    {
        cout<< "Overloading delete operator " << endl;
        free(p);
    }
};

int main()
{
    student * p = new student("NIKHIL", 20);

    p->display();
    delete p;
}
```

OUTPUT :

```
Overloading new operator with size: 16
Name:NIKHIL
Age:20
Overloading delete operator


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 23.

AIM : Write a program to overload unary minus(-) operator using friend function.

THEORY :

Unary operator acts on one operand only. In case overloaded operator function is a class member function, then it will act on the object with which it is called and use it as operand. Hence we need not to pass any extra argument in unary operator function if its class member function.

Let's see how to overload Unary Minus (–) operator for above class i.e.

```
/*
* Overloaded unary minus operator as member function.
* It returns a new Object.
*/
ComplexNumber ComplexNumber::operator-() const
{
return ComplexNumber(-(this->real), -(this->imaginary) );
}
```

It returns a new object with modified values.

CODE :

```cpp
#include<iostream>

using namespace std;

class A
{
private:
int a;

public:
void set_a();
void get_a();
friend A operator -(A); // Friend function which takes an object of A and return an object of A type.
};


//Definition of set_a() function
void A :: set_a()
{
a = 10;
}


//Definition of get_a() function
void A :: get_a()
{
cout<< a <<"\n";
}


//Definition of overloaded unary minus operator - friend function
A operator -(A ob)
{
ob.a = -(ob.a);
return ob;
}


int main()
{
A ob;
ob.set_a();

cout<<"The value of a is : ";
ob.get_a();

//Calling operator overloaded function - to negate the value
ob = -ob;   //ob object is passed as an argument to the friend function and its negative version is returned.

cout<<"The value of a after calling operator overloading friend function - is : ";
ob.get_a();
}
```

OUTPUT :

```
The value of a is : 10
The value of a after calling operator overloading friend function - is : -10


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 24.

AIM : Create a base class basic_info with data members name ,roll no, sex and two member functions getdata and display. Derive a class physical_fit from basic_info which has data members height and weight and member functions getdata and display. Display all the information using the object of the derived class.

THEORY :

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.
Let us take previously defined class to access the members of the class using a member function instead of directly accessing them −

```
class Box
{
  public:
    double length;         // Length of a box
    double breadth;        // Breadth of a box
    double height;         // Height of a box
    double getVolume(void);// Returns box volume
};
```

Member functions can be defined within the class definition or separately using scope resolution operator, : −. Defining a member function within the class definition declares the function inline, even if you do not use the inline specifier.

So either you can define Volume() function as below −

```
class Box
{
```

```
  public:
     double length;      // Length of a box
     double breadth;     // Breadth of a box
     double height;      // Height of a box

     double getVolume(void)
     {
       return length * breadth * height;
     }
};
```

If you like, you can define the same function outside the class using the scope resolution operator (::) as follows −

```
double Box::getVolume(void)
{
  return length * breadth * height;
}
```

Here, only important point is that you would have to use class name just before :: operator. A member function will be called using a dot operator (.) on a object where it will manipulate data related to that object only as follows −

```
Box myBox;          // Create an object

myBox.getVolume();  // Call member function for the object
```

CODE :

```cpp
#include<iostream>

using namespace std;

const int MAX = 100;

class basic_info
{
private:
  char name[25];
  int rollno;
  char sex;
public:
  void getdata1();
  void display1();
};

class physical_fit:public basic_info
{
private:
  float height;
  float weight;
public:
  void getdata();
  void display();
};

void basic_info::getdata1()
{
  cout << "Enter a name: ";
  cin >> name;
  cout << "Roll no: ";
  cin >> rollno;
  cout << "Sex: ";
  cin >> sex;
}

void basic_info::display1()
{
  cout << name << "\t";
  cout << rollno << "\t";
  cout << sex << "\t";
}

void physical_fit::getdata()
{
  getdata1();
  cout << "Height(m): ";
  cin >> height;
  cout << "Weight(Kg): ";
  cin >> weight;
}

void physical_fit::display ()
{
  display1();
```

```
  cout << height << "\t";
  cout << weight << " ";
}

int main()
{
  physical_fit a[MAX];
  int i, n;
  cout << "How many students: ";
  cin >> n;
  cout << "Enter the following infromation\n";
  for (i = 0; i < n; i++)
    {
      cout << "Record: " << i + 1 << endl;
      a[i].getdata ();
    }
  cout << endl;
  cout << "_____\n";
  cout << "Name\tRollNo\tSex\tHeight\tWeight\n";
  cout << "_____\n";
  for (i = 0; i < n; i++)
    {
      a[i].display ();
      cout << endl;
    }
  cout << endl;
  cout << "_____\n";
  return 0;
}
```

OUTPUT :

```
How many students: 2
Enter the following infromation
Record: 1
Enter a name: NIKHIL
Roll no: 4
Sex: M
Height(m): 160
Weight(Kg): 60
Record: 2
Enter a name: ROHIT
Roll no: 5
Sex: M
Height(m): 170
Weight(Kg): 70


_____
Name    RollNo  Sex     Height  Weight
_____
NIKHIL  4       M       160     60
ROHIT   5       M       170     70


_____


...Program finished with exit code 0
Press ENTER to exit console.
```

LAB-8

07-06-21

EXPERIMENT 25.

AIM : Create class first with data members book no, book name and member function getdata() and putdata(). Create a class second with data members author name, publisher and members getdata() and showdata(). Derive a class third from first and second with data member no of pages and year of publication. Display all this information using an array of objects of the third class.

THEORY :

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.
Let us take previously defined class to access the members of the class using a member function instead of directly accessing them −

```
class Box
{
  public:
    double length;         // Length of a box
    double breadth;        // Breadth of a box
    double height;         // Height of a box
    double getVolume(void);// Returns box volume
};
```

Member functions can be defined within the class definition or separately using scope resolution operator, : −. Defining a member function within the class definition declares the function inline, even if you do not use the inline specifier.

So either you can define Volume() function as below −

```
class Box
{
  public:
```

```
   double length;      // Length of a box
   double breadth;     // Breadth of a box
   double height;      // Height of a box

   double getVolume(void)
   {
     return length * breadth * height;
   }
};
```

If you like, you can define the same function outside the class using the scope resolution operator (::) as follows −

```
double Box::getVolume(void)
{
  return length * breadth * height;
}
```

Here, only important point is that you would have to use class name just before :: operator. A member function will be called using a dot operator (.) on a object where it will manipulate data related to that object only as follows −

```
Box myBox;          // Create an object

myBox.getVolume();  // Call member function for the object
```

CODE :

```
#include <iostream>

using namespace std;

class first
{
  char name[20];
    int bookno;
public:
  void getdata ()
  {

    cout << "\nEnter the name of book : ";
    cin >> name;
    cout << "\nEnter Book No. : ";
    cin >> bookno;
  }
  void putdata ()
  {
    cout << "\nName of Book : " << name;
    cout << "\nBookNo. :" << bookno;
  }
};

class second
{
  char author[20], publisher[20];
public:
  void getdata ()
  {
    cout << "\nEnter Author's Name : ";
    cin >> author;
    cout << "\nEnterPublisher : ";
    cin >> publisher;
  }
  void showdata ()
  {
    cout << "\nAuthor'sName : " << author;
    cout << "\nPublisher : " << publisher;
  }
};

class third:public first, public second
{
  int no_pages, year;
public:
  void get ()
  {
    first::getdata ();
    second::getdata ();
    cout << "\nEnter No. of Pages : ";
    cin >> no_pages;
    cout << "\nEnter the year of publication : ";
    cin >> year;
  }
  void display ()
```

```
  {
    putdata ();
    showdata ();
    cout << "\nNo. of Pages : " << no_pages;
    cout << "\nYear of Publication : " << year;
  }
};

int main()
{
  third book[5];
  int num;
  cout << "\nEnter the number of books : ";
  cin >> num;
  for (int i = 0; i < num; i++)
    {
      book[i].get ();
      cout << endl;
    }
  for (int i = 0; i < num; i++)
    {
      book[i].display ();
      cout << endl;
    }
}
```

OUTPUT :

```
Enter No. of Pages : 177

Enter the year of publication : 2021


Enter the name of book : MUNCH

Enter Book No. : 5

Enter Author's Name : ROHIT

EnterPublisher : SG

Enter No. of Pages : 771

Enter the year of publication : 2021


Name of Book : KITKAT
BookNo. :4
Author'sName : NIKHIL
Publisher : GS
No. of Pages : 177
Year of Publication : 2021

Name of Book : MUNCH
BookNo. :5
Author'sName : ROHIT
Publisher : SG
No. of Pages : 771
Year of Publication : 2021


...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 26.

AIM : Design three classes STUDENT , EXAM and RESULT. The STUDENT class has data members such as rollno, name. create a class EXAM by inheriting the STUDENT class. The EXAM class adds data members representing the marks scored in six subjects. Derive the RESULT from the EXAM class and has its own data members such as total marks. Write a program to model this relationship.

THEORY :

C++ Classes :

Class: A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc.
So here, Car is the class and wheels, speed limits, mileage are their properties.

1. A Class is a user defined data-type which has data members and member functions.
2. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
3. In the above example of class Car, the data member will be speed limit, mileage etc and member functions can be apply brakes, increase speed etc.

CODE :

```cpp
#include <iostream>
using namespace std;

class Student{
    char name[50];
    int roll_no;
  public:
    void getData(){
        cout<<"Enter Name: "; cin>>name;
        cout<<"Enter Roll No: "; cin>>roll_no;
    }
    void putData(){
        cout<<"\n\nName: "<<name;
        cout<<"\nRoll No: "<<roll_no;
    }
};

class Exam : public Student{
    protected:
        int marks[6];
    public:
        void getData();
        void putData();
};

void Exam::getData(){
    cout<<"Enter marks in 6 subjects: ";
    for(int i=0;i<6;i++) cin>>marks[i];
}

void Exam::putData(){
    cout<<"\nMarks in 6 subjects: ";
    for(int i=0;i<6;i++) cout<<marks[i]<<" ";
}

class Result : public Exam{
    int total_marks;
  public:
    void calculate();
    void putData(){
        cout<<"\nTotal Marks: "<<total_marks;
    }
};

void Result::calculate(){
    total_marks = 0;
    for(int i=0;i<6;i++) total_marks+= marks[i];
}
int main(){
    Result obj;
    obj.Student::getData();
    obj.Exam::getData();
    obj.calculate();
    obj.Student::putData();
    obj.Exam::putData();
    obj.putData();
    return 0;
}
```

OUTPUT :

```
Enter Name: NIKHIL
Enter Roll No: 1
Enter marks in 6 subjects: 70
70
70
70
70
70


Name: NIKHIL
Roll No: 1
Marks in 6 subjects: 70 70 70 70 70 70
Total Marks: 420

...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 27.

AIM : Create a base class called SHAPE. Use this class to store two double type values. Derive two specific classes called TRIANGLE and RECTANGLE from the base class. Add to the base class, a member function getdata to initialize base class datamembers and another member function display to compute and display the area of figures. Make display a virtual function and redefine this function in the derived classes to suit their requirements. Using these three classes design a program that will accept driven of a TRINGLE or RECTANGLE interactively and display the area.

THEORY :

C++ Classes :

Class: A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc.
So here, Car is the class and wheels, speed limits, mileage are their properties.

4. A Class is a user defined data-type which has data members and member functions.
5. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

In the above example of class Car, the data member will be speed limit, mileage etc and member functions can be apply brakes, increase speed etc.

CODE :

```cpp
#include <iostream>
using namespace std;

class Shape{
    protected:
        double length;
        double breadth;
    public:
        void getData(){
            cout<<" Enter length: "; cin>>length;
            cout<<" Enter breadth: "; cin>>breadth;
        }
        virtual void display() = 0;
};

class Triangle : public Shape {
    public:
        void display(){
            double area = 0.5*length*breadth;
            cout<<"\n\n Area of triangle: "<<area;
        }
};

class Rectangle : public Shape {
    public:
        void display(){
            double area = length*breadth;
            cout<<"\n\n Area of rectangle: "<<area;
        }
};

int main(){
    cout<<"\n\nCalculate area: \n 1. Triangle\n 2. Rectangle\n\n Enter choice: ";
    int choice; cin>>choice;
    switch(choice){
        case 1: {
            Triangle triangle;
            triangle.Shape::getData();
            triangle.display(); break;
        }
        case 2: {
            Rectangle rectangle;
            rectangle.Shape::getData();
            rectangle.display(); break;
        }
        default: cout<<" Wrong choice!";
    }
    return 0;
}
```

OUTPUT :

```
Calculate area:
 1. Triangle
 2. Rectangle

 Enter choice: 2
 Enter length: 80
 Enter breadth: 10


 Area of rectangle: 800

...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT 28.

AIM : Create a class called LIST with two pure virtual functions store() and retrieve(). To store a value call store and to retrieve call retrieve function, Derive two classes stack and queue from it and override store and retrieve.

THEORY :

C++ Classes :

Class: A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc.
So here, Car is the class and wheels, speed limits, mileage are their properties.

6. A Class is a user defined data-type which has data members and member functions.
7. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

In the above example of class Car, the data member will be speed limit, mileage etc and member functions can be apply brakes, increase speed etc.

CODE :

```
#include <iostream>
using namespace std;

class List{
 public:
    virtual void store(int value) = 0;
    virtual int retrieve() = 0;
};

class Stack:public List{
    int values[50];
    int top;
 public:
    Stack(){
        top=-1;
    }
    void store(int value){
        if(top++>=50){
            cout<<"Overflow!!";
            --top;
            return;
        }
        values[top] = value;
    }

    int retrieve(){
        if(top==-1){
            cout<<"Underflow";
            return -1;
        }
        return values[top--];
    }
    void display();
};

void Stack::display(){
    int i = 0;
    while(i<=top){
```

```
39          cout<<values[i]<<", ";
40          ++i;
41      }
42  }
43
44  class Queue:public List{
45      int values[50];
46      int front,rear;
47   public:
48      Queue(){
49          front=0;
50          rear=0;
51      }
52      void store(int value){
53          if((rear+1)==front){
54              cout<<"Overflow";
55              return;
56          }
57          values[rear] = value;
58          rear++;
59          rear=rear%50;
60      }
61
62      int retrieve(){
63          if(rear==front){
64              cout<<"Underflow";
65              return -1;
66          }
67          if(front==49){
68              front=0;
69              return values[49];
70          }
71          return values[front++];
72      }
73      void display();
74  };
75
76  void Queue::display(){
77       int i=front;
78       while(i<rear){
79           cout<<values[i]<<", ";
80           i++;
81           i%=50;
82       }
83       cout<<endl;
84  }
```

```cpp

int main(){
    Stack stack;
    Queue queue;
    cout<<" MENU\n1. Add to stack\n2. Add to Queue\n3. Delete from Stack\n4. Delete from Queue\n5. Display stack and queue\n6. Exit\n";
    while(1){
        cout<<"\n Enter your choice: ";
        int choice;
        cin>>choice;
        switch(choice){
            case 1: int value;
                cout<<"\n Enter value: ";
                cin>>value;
                stack.store(value);
                cout<<" "<<value<<" Added to stack\n";
                break;
            case 2: cout<<"\n Enter value: ";
                cin>>value;
                queue.store(value);
                cout<<endl<<" "<<value<<" Added to queue\n";
                break;
            case 3: cout<<" "<<stack.retrieve()<<" Deleted from stack\n";
                break;
            case 4: cout<<" "<<queue.retrieve()<<" deleted from queue\n";
                break;
            case 5: cout<<"\n Stack: ";
                stack.display();
                cout<<"\n Queue: ";
                queue.display();
                break;
            case 6: exit(1);
                default: cout<<" Wrong choice!!";
        }
    }
    return 0;
}
```

OUTPUT :

```
 MENU
1. Add to stack
2. Add to Queue
3. Delete from Stack
4. Delete from Queue
5. Display stack and queue
6. Exit

 Enter your choice: 1

 Enter value: 2
 2 Added to stack

 Enter your choice: 1

 Enter value: 4
 4 Added to stack

 Enter your choice: 5

 Stack: 2, 4,
 Queue:

 Enter your choice: 3
 4 Deleted from stack

 Enter your choice: 5

 Stack: 2,
 Queue:

 Enter your choice: 6


...Program finished with exit code 0
Press ENTER to exit console.
```